1-1-2002

Intrusion detection through knowledge sharing

Christopher T. Kirk
Iowa State University

Intrusion detection through knowledge sharing

by

Christopher T. Kirk

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Engineering

Program of Study Committee:
Julie Dickerson (Major Professor)
James Davis
Johnny Wong

Iowa State University

Ames, Iowa

2002

Graduate College

Iowa State University

This is to certify that the master's thesis of

Christopher T. Kirk

has met the thesis requirements of Iowa State University

Signatures have been redacted for privacy

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

I would like to thank my major professor, Dr. Julie Dickerson, for her time, patience, and support during my stay at Iowa State. I would also like to thank John Dickerson and Dr. Jim Davis for getting me started on my research. I am also grateful for Dr. Johnny Wong's reminder that information is not the same as knowledge: that nugget of wisdom helped me make my writing more legible.

Above all else I must thank my friends, especially Aaron Peden and Joy Nelson, without whom this paper would contain 178 more commas than it already does. Thank you, friends, for keeping me sane as I wrote this thesis and for believing in me.

Last but not least, I thank the discovery of the caffeine molecule. Without caffeine none of this would *ever* have been possible.

INTRODUCTION

Computer vulnerabilities occur so frequently that they usually are not even newsworthy. The FBI report on computer intrusions for 2000 lists over $260 million in damages, stolen goods, and lost profits [Scambray et al, 2001]. Computer systems are being hardened against attacks and more money is being spent on computer security, but several new security threats are still being discovered every month. Company policies that balance convenience against security prevent the creation of a 'completely secure' system [Lee and Stolfo, 2002]. In fact, it is unlikely that any computer system could *ever* be 'completely secure.'

Companies attempt to prevent as many intrusions as possible, but it is unreasonable to think that all intrusions can be stopped. However, the purpose of the Intrusion Detection System (IDS) is not actually to stop intrusions: an IDS alerts system administrators of successful intrusions so that companies can attempt to minimize their losses. Additionally, an IDS reveals new computer vulnerabilities and identifies the areas where security should be increased. Successful intrusion detection systems reduce the risk, and cost, of an electronic attack. Additional intrusion detection capability is added to a network only if its additional risk reduction justifies the extra cost [Porras and Valdes, 1997]. There are added risks involved with replacing an already-functional intrusion detection system. People are also inherently suspicious of the security of any new software [Nikander and Karvonen, 2001]. Due to the risks and suspicions, many companies require a new IDS to prove itself significantly better than older systems before they consider implementing it.

Most intrusion detection methods can be classified as either 'misuse detection' or 'anomaly detection.' Misuse detection systems compare current network events to a list of signatures associated with known attacks. Anomaly detection systems calculate statistical estimates of 'normal' events and declare abnormal inputs as intrusions. More details about misuse and anomaly detection are included later in this thesis.

It is difficult to compare two intrusion detection systems and determine which one is 'better' than the other. Each IDS uses different sources and methods of analyzing data. System detection performance varies according to the testing conditions. In order to interpret the results of IDS comparison tests, it is important to understand the strengths and weaknesses of IDS testing.

Current intrusion detection sensors can be thought of as functioning in a 'dumb' mode. Immediately after detecting an attack, an alert is transmitted and effectively forgotten by the detecting sensor. Knowledge propagates to the system administrators responsible for security, but the intrusion detection system does not retain and use the knowledge. A better solution would be to implement a method of sharing knowledge that allows an IDS to take advantage of knowledge after it is learned. Better still is a knowledge-sharing scheme that can apply to existing intrusion detection products. If companies could apply new research to their existing intrusion detection systems without unnecessarily high risk or cost, then the research is more likely to be used.

Previous research has not centered on cost effective ways to apply new intrusion detection methods to existing systems. Instead, the past research has focused primarily on new intrusion detection algorithms and new ways to reduce the quantity of input data. This research emphasizes anomaly detection systems because they show more potential than

misuse detection systems. Some research investigates the possibility of hybrid systems that combine elements of both misuse and anomaly detection systems [Neumann and Porras, 1999]. The underlying theme in the majority of this research is the development of a new intrusion detection tool or product. Researchers do not discuss integrating the new methods or new products with existing intrusion detection systems. Few research papers explore the concept of cooperation between IDS components.

This thesis outlines a new knowledge-sharing concept and IDS protocol. This concept is flexible enough to allow for the integration of already existing IDS sensors that were not designed with knowledge sharing in mind. The research addressed within this thesis demonstrates the increased usefulness of simple sensors by sharing and using knowledge of attacks as it is learned.

STAGES OF A SOPHISTICATED ATTACK

The greatest danger to a properly-secured network is not a 'script kiddy' mindlessly searching the Internet for old published exploits, but the sophisticated attacker who puts time and effort into finding and exploiting security weaknesses. Regardless of the attacker, the actions needed to break into a computer can be broken into five very specific phases: Reconnaissance, Information Gathering, Vulnerability Exploitation, Backdoor Installation, and finally Information Leakage/Theft of Resources [Scambray et al, 2001]. Knowing what the hacker needs to do is vital to understanding how intrusion detection systems must operate. Therefore, these phases shall be described in detail below. Information is included about what the attacker learns, and what information a careful system administrator can learn, during the attack phase.

Stage One: Reconnaissance

The first stage of a sophisticated attack is always reconnaissance: a hacker cannot attack a computer until he or she knows the computer exists. Reconnaissance consists of three actions: ping scans, port scans, and service scans. Ping scans sweep through IP addresses to find out which addresses are currently in use, and which are not. The use of DSL and cable modems makes home computers more vulnerable than the days of dial-up, because the computers are now accessible via Internet 24 hours a day, instead of only for the times when the computer is in use. A port scan sweeps through all TCP and UDP ports on one IP address to check and see which ports are listening. This scan allows hackers to find services to use in breaking into the computer. Service scans function by searching a range of

computers, whose IP addresses were gathered from an earlier ping scan. These service scans do not search all available ports, instead they search only pre-selected and commonly used ports, such as port 23 for telnet, or port 80 for web traffic. Service scans are popular shortly after a new exploit is released, by hackers not interested in a full-scale intrusion attempt, but rather are looking for easily compromised computers [Northcutt and Novak, 2000]. Hackers need not use all three methods of reconnaissance, though reconnaissance of some form is required to find systems to target.

Through reconnaissance, a hacker learns the IP address of all accessible systems. Though there are likely to be more computers hidden behind a firewall, the hacker now has a list of computers that might be vulnerable from his or her current point of attack. The hacker also learns which IP addresses belong to servers, and what services are running on those servers. Now the hacker knows where to focus his or her attention, and can begin to gather more detailed information about the running services.

Reconnaissance scans are common events in a system administrator's life, and they are completely legal. But, while a sys-admin cannot punish would-be intruders who scan their networks, there is information to be gained by studying the reconnaissance methods used. Reconnaissance works as an early-warning method for intrusion detection. The originating IP address is clearly within the control of a potential hacker, and further events from that IP address warrants additional scrutiny. Any active IP address or service port not scanned by the hacker is less likely to come under further attack. In addition, if the hacker performs service scans, then the hacker just informed the system administrator as to the nature of the tools at the hacker's disposal. Repeated service scans should prompt a review of recent vulnerability announcements, and an increased focus of security on the scanned

ports. Scans occur all the time and not all scans lead to serious attack threats. However, system administrators are better off erring on the side of caution as much as is practical.

Stage Two: Information Gathering

The reconnaissance stage provides important general information, but lacks the details a hacker requires to compromise a computer network. After reconnaissance, a hacker's next stage of attack is an in-depth Information Gathering phase. During the Information Gathering stage, the objective is to learn as much as possible about the target network, including:

- Network Topology
- Firewall Rules
- Computer Operating Systems
- Software Version Information
- Server Software Versions
- User Accounts
- Public File Shares

This information lets a hacker search for applicable exploits, devise ways to guess passwords, and bypass firewall security [Anonymous, 1998].

In this phase of the attack a hacker can expect to learn, at the least, the operating system and OS version of every computer scanned, as well as vendor and version information for most of the common server software [Scambray et al, 2001]. How much

more information a hacker learns depends largely on his or her tenacity, and on the security configuration of the target network.

Like reconnaissance, information gathering is generally legal, so system administrators are limited in their ability to stop hackers at this phase in the attack. However, it is important to study what information a hacker attempts to gather. At this stage, a system administrator with capable intrusion detection should be able to create a list of the information that the hacker has learned. A successful intrusion cannot occur without this information, so the systems most vulnerable to the hacker are the ones about which he has gathered this information. An intrusion detection system could be configured to weight unusual activity from the hacker's known IP address(es) if the hacker's target is one of the services or machines previously targeted for in-depth information gathering.

Stage Three: Vulnerability Exploitation and Privilege Escalation

After gathering as much information as possible, a hacker will have a good idea of the weakest points in the target network's security. The attacker then moves on to stage three by exploiting one of the discovered remote vulnerabilities. A remote vulnerability is simply any vulnerability that can be used by a hacker who does not already have the ability to log into a computer [Anonymous, 1998]. One of the 14 most commonly exploited vulnerabilities is a weak password [Scambray et al, 2001]: a user password easily guessable given the information known to a hacker. Recently, buffer overflow exploits have become common, as they are effective in scripts and require less effort on the hacker's behalf.

After vulnerability exploitation, a hacker has either shell access, or a more limited ability to execute code of his or her choice. This generally won't be enough to give root

access to a computer, and a hacker is rarely satisfied with anything less. So, the next goal of the sophisticated hacker is privilege escalation: taking the access they do have, and finding another vulnerability that will give root access. Privilege escalation is easier than it seems. At this point the hackers have access to more commands and more vulnerable software as they have assumed the identity of a user. Any local vulnerability that can lead to root access will suffice, and there are many local programs that require some degree of privileged access to function.

The vulnerability exploitation stage is the most difficult step to detect, and since this is the stage where hackers first begin to execute code of their choice, the vulnerability exploitation stage is arguably the most important stage *to* detect [Ruiu, 2001]. If the hacker is detected at this point, a system administrator can learn information valuable to the entire community of security experts: namely, one can learn about the exploits that exist in the wild, and are currently being used by one or more hackers [Northcutt and Novak, 2000]. A system administrator can also learn which services, computers, and user accounts are vulnerable to attack. Armed with this information, more effective defenses can be created for less cost.

Stage Four: Backdoor Installation

Once in full control of a computer, a hacker's next goal is to ensure full access to this computer in the future. Stage four, the installation of a backdoor, is a hacker's method of protecting the time invested in hacking a computer by allowing an easier method of gaining access in the future. This is important, since eventually the vulnerability that was used will be patched. Once that happens, a new vulnerability would have to be found on any system

without a backdoor installed. The three most common methods of installing a backdoor are: super-user account creation, attaching command shells to TCP ports, and 'root kits.'

The simplest backdoor is to create an additional user account, usually with extra privileges. In a computer network with many computers and user accounts, detecting one more account can be a daunting task. Since this method only works well with computers running some form of remote shell access (ssh and telnet, for example), more complicated backdoors were created.

For computers that do not already have remote shell access, dozens of tools exist to help create command-shell backdoors. If telnet or ssh server software already exists on the computer, a hacker merely needs to configure the server to run on an unusual port number that isn't likely to be noticed. Otherwise, software like netcat (http://sources.isc.org/network/status/netcat-110.txt) can be used to bind a command shell's I/O to a TCP port. Some of these backdoor utilities, such as Loki (http://www.phrack.com/show.php?p=49&a=6), even provide capabilities for passing through firewalls.

The most complicated of the backdoors is the 'root kit' [Scambray et al, 2001]. A root kit is an entire suite of programs and utilities designed to make a hacker's job as convenient as possible. Root kits come with modified copies of commonly run utilities, designed to prevent detection of the backdoor. Some root kits even come with stealthy command shells, which are harder for system administrators to detect.

Most hackers reuse the same types of backdoors, so when an intrusion detection system reports the presence of a backdoor, the system administrator knows to search the entire network for other instances of the same backdoor. The sys-admin also learns an

important fact: that a computer has been compromised. Knowing precisely which computer was compromised, and having a rough time frame to work with, a system administrator can then tune intrusion detection software to scan through old logs for unusual activity involving the hacker's IP address(es) and the compromised computer.

Stage Five: Information Leakage and Theft of Resources

The final stage of an attack represents the ultimate goal of the hacker: putting the new gains to use. Hackers often break into computers just for the challenge or the fun of it. Regardless, all compromised computers are eventually put to use by the hacker. The Information Leakage and Theft of Resources stage represents the last stage of a sophisticated attack. In this stage, a hacker can do two things: the first is to steal information such as credit card information, software source code, and corporate secrets [Scambray, 2001]. The second option for a hacker is to steal computer resources for personal gain. Many common uses include illegal software storage, IRC servers, cracking password files to other computers, and using the computer as a stepping-stone to attack additional systems [Ruiu, 2001].

During this process, a hacker can examine all of the data stored on the computer. The hacker may also learn about other computers in the same network, and computers that communicate or perhaps have trusted status with the newly hacked computer. With a small amount of effort, hackers can gain a new list of IP addresses to target, as well as potential usernames and passwords.

A system administrator who did not catch the intrusion before this stage can still detect theft of resources. The information leakage can be monitored by study of unusual data

transfers. A common example would be a sudden surge in outbound traffic load to an off-site IP address. Other resource theft can be detected by looking for missing disk space and analyzing CPU usage for unusual running processes. Once detected, a system administrator can learn two facts about the hacker that can help in intrusion detection. The first and most obvious piece of information learned is the IP addresses of computers controlled by the hacker. The second is an idea of when the hacker is active, though this information often takes more time to learn than system administrators are comfortable giving hackers. With one or both of these facts, old log files can be readily filtered to detect previously missed signs of intrusion, and to detect future signs of intrusion that might otherwise be overlooked.

GENERAL INTRUSION DETECTION CONCEPTS

Intrusion detection is not a new field of study. As such, several basic aspects of the field are well understood by security personnel and by most system administrators. The three basic measures of an intrusion detection system (detection rate, false alarm rate, and detection speed) are often used as a selling point of commercial systems. The three most common sources of data for intrusion detection (network traffic logs, computer audit logs, and service audit logs) are also well known, as are the basic principles of misuse-detection and anomaly-detection methods. Since people often use different terminology while meaning the same idea, an explanation for these terms follows.

The three most commonly described goodness measures of an intrusion detection system are: detection rate, false alarm rate, and detection speed [Ye et al, 2001]. The detection rate measures the percentage of intrusions detected by the IDS. In order for the IDS to be effective this rate must be close to one hundred percent: any IDS with a detection rate of less than eighty percent is probably of no use. The second goodness measure, called the false alarm rate, measures the percentage of 'normal' data samples that are incorrectly labeled as intrusions. The third goodness measure, detection speed, describes the delay between an intrusion and its detection. Ideally an IDS should report intrusions in real-time, but this is rarely feasible because of the computational time involved.

These factors do not give a complete picture of the performance of an intrusion detection system. Another factor is the amount of detail an IDS can provide. Merely knowing an attack occurred at noon is not enough to defend against it: a system administrator must know something about the attack itself. Another overlooked factor is the

ease of using the IDS [Northcutt and Novak, 2000]. The better the organization, and the easier to read an intrusion detection system's reports are, the more efficiently administrators can use this information.

Data Types Available

In order to study and design intrusion detection systems, it is necessary to understand the information available for use. With the ever-growing increase of traffic on Internet and intranets alike, there is no shortage of data for intrusion detection. There are too many potential data sources to discuss each in depth, so instead the data sources will be discussed by category. The three main categories of intrusion detection data are: network traffic logs, computer audit logs, and service audit logs. A brief description of each category, as well as their strengths and weaknesses, follows.

Perhaps the most commonly used source of data is a network traffic log. These logs rely on a program, such as Tcpdump (http://www.tcpdump.org), to copy all of the packets on the network to a log file. Network traffic logs have two primary advantages. First, they have the ability to monitor across many computers at once, rather than requiring a separate intrusion detection sensor per computer. The second advantage is that network traffic logs contain information on every packet sent or received on the network: this allows the logs to determine most security-related events that occur on a network. Unfortunately, the extremely high volume of network traffic means that Tcpdump and similar programs cannot store the payload of packets, merely the header information. Thus, TCP connections themselves can be monitored, but the data and user commands sent over the connections cannot [Mahony and Chan, 2001].

The second category of data is the computer audit logs. The file names and the exact content of the files will vary according to platform, but remain the same in principle: computer audit logs store information about unexplained errors, anomalous events, and attempted security violations. The computer audit logs record important state information of the operating system and system calls [Lee and Stolfo, 2002]. The audit log files can collect information about local attacks originating from the physical console of a computer [Ye et al, 2001]. No other category of intrusion detection data works against such an attack. This fact makes computer audit logs immensely useful in protecting computer desktops, where console access is commonplace. It also makes computer audit logs less useful in protecting servers, where almost all traffic is remote-based. One of the disadvantages with intrusion detection systems that only use computer audit logs is that they require one IDS sensor per computer. Another disadvantage is that a different IDS sensor must be developed for each operating system, because each operating system reports different information and in different ways.

The third and final major category of intrusion detection data comes from service audit logs. These log files are very similar to computer audit logs, except they are generated by server programs, such as Apache's web server (http://httpd.apache.org). The detection scopes of such log files are extremely limited, but the amount of detail provided to the intrusion detection system increases as the detection scope narrows. Service audit logs have the distinct advantage of being the only source of data that describes the internal state of server software. This information can be invaluable for intrusion detection. Unfortunately the quality of the data collected depends upon the code of other developers, which often prevents this data source from being as useful as it could be. Another disadvantage of service audit logs is that they only augment intrusion detection - these log files alone cannot

provide comprehensive intrusion detection capabilities. Given these three types of detection data, the next step is to understand the basic types of detection systems.

Categories of Intrusion Detection Systems

Intrusion detection systems are usually categorized by the principle they use for detecting attacks. This works well, as so far only two principles exist: "misuse detection" and "anomaly detection." A third category has appeared relatively recently in literature: hybrid-class systems, which combine some of the best features of both detection principles.

Misuse detection systems take a reactive approach to computer intrusions. Every known intrusion technique is given an identifying signature. By using identifying signatures, a misuse-based IDS detects any future use of these known intrusion methods [Porras and Valdes, 1997]. The misuse approach provides high detection rates with known intrusion methods, but cannot detect novel intrusion methods. The misuse approach also provides low false alarm rates. Additionally, many misuse-based intrusion detection systems, such as Snort (http://www.snort.org) and Network Flight Recorder (http://www.nfr.com), work in real-time. The pattern-recognition approach forces malicious intruders to constantly find new ways to break into a computer network, but pattern-recognition systems cannot detect, prevent, or minimize the damage from these novel network intrusions. The shortcomings of misuse detection systems prompted the development of the more proactive anomaly detection systems.

Anomaly-based intrusion detection systems assume that any event outside the range of normal behavior must be an intrusion [Denning, 1987]. Every user, program, server, or printer will have a normal behavioral profile. Successful security penetrations happen rarely

on most secure networks, and therefore count as abnormal events. It is reasonable to assume that abnormal events can be reported as intrusions. However, an anomaly-based IDS will have a higher false-alarm rate than a misuse-based system. Frequently, factors such as commercial advertising and unexpected business deadlines cause temporary changes in normal behavior. An anomaly-based system would likely misinterpret these changes as intrusions. This class of IDS does more than merely compare input data with pre-specified patterns, and therefore requires more processing power than the misuse detection class. While misuse-based systems can report in real-time, most anomaly-based systems report at least a few minutes after intrusions. Also, buffer-overflow attacks, which are often a single TCP packet, are frequently undetectable by anomaly-based systems. Even with these problems, anomaly-based systems are capable of detecting intrusions that have never been seen before, and produce higher detection rates than misuse-detection systems. Therefore, this class of IDS has an advantage over misuse-based systems, and anomaly-detection systems are beginning to replace misuse-based systems.

Recently, hybrid intrusion detection systems have become more popular in research papers. Hybrid systems attempt to combine the best features of both misuse-based and anomaly-based intrusion detection systems. Because of their increased complexity, hybrid systems tend to have more difficulty than other classes in the areas of data reduction and information sharing. One of the most well-known hybrid systems is EMERALD (Event Monitoring Enabling Responses to Anomalous Live Disturbances) [Porras and Valdes, 1997].

Since EMERALD is perhaps the most highly visible of the hybrid intrusion detection systems, a brief description of the product is included to provide a general overview of the

class as a whole. EMERALD is a multiple-sensor system, with separate misuse-based and anomaly-based detection mechanisms [Neumann and Porras, 1999]. The output of these differing sensors becomes combined at one site (probably a computer dedicated to the task), where the incoming intrusion detection data is filtered, analyzed, compacted, and reported to system administrators monitoring the network. EMERALD combines both sensor types because anomaly detection is rarely detailed and reliable enough when reporting known attacks, and misuse detection by itself is simply not sufficient:

> It should be no surprise to those in the intrusion-detection community that signature-based analysis is good at detecting and identifying well-defined known scenarios, but very limited in detecting hitherto unknown attacks (except for those that happen to trigger existing rules serendipitously). On the other hand, statistical profile-based analysis can be effective in detecting unknown attacks and providing early warnings on strangely deviant behaviors; however, the statistical approach does not naturally contribute to an automated identification and diagnosis of the nature of an attack or other type of deviation that it has never identified before. Although inferences can be drawn about the nature of an anomaly, based on the statistical measures that were triggered, further reasoning is typically necessary to identify the nature of the anomaly...[Neumann and Porris, 1999]

Hybrid systems have all the strengths and few of the weaknesses of the two basic intrusion detection systems. Hybrid systems are capable of detecting the same attacks in real-time that misuse-detection systems find, and hybrids can detect buffer-overflow attacks with the same pinpoint accuracy as a misuse-detection system. Additionally, by controlling the relative importance of the two detection methods, misuse and anomaly, one controls the ratio of detections rate to false alarm rate. Hybrid systems pay for this with added computational cost, however.

To demonstrate the added data reduction difficulty that hybrid systems deal with, consider a hybrid IDS composed of two sensors: one misuse-based, and one anomaly-based. A hybrid intrusion detection system must deal with the same data reduction problem either category of IDS faces individually, and must also deal with combining the output of both sensors into a single meaningful result. The data reduction problem will be addressed in more detail in the experimental results section of the thesis.

To achieve higher detection rates and lower false alarm rates, a hybrid system must not just take the average result of several existing systems, but must instead share meaningful information between sensors with different capabilities. Any multi-sensor intrusion detection system must deal with the information-sharing problem, but hybrid systems are more likely to have multiple types of sensors. Since these sensors would produce different output than the other sensors, or require different input data to function, the information-sharing problem becomes more complicated for the hybrid class. The result is that while the anomaly-based IDS become more popular commercially, the hybrid systems remain limited mostly to academia and companies with very large budgets.

METHODS ON COMPARING IDS SYSTEMS

Once the field of intrusion detection had evolved to the point of having multiple commercially available systems, consumers have sought to know which product is the "best." Comparing two systems is much the same as comparing two automobiles: everyone uses different criteria in determining which is best. Over the years, IDS comparisons have become more accurate, more impartial, and more realistic, but the IDS tests still need to evolve further. A brief history of intrusion detection system comparisons follows, with a description of their pros and cons, and a series of guidelines for designing better IDS tests in the future.

History of IDS Comparisons

Originally each vendor would test their own intrusion detection products, and report back statistics such as detection rate, false alarm rate, and the processing time required for detection. These are the most important considerations for an IDS but the original tests lacked credibility. Each vendor tested their own products, making the integrity of the results questionable, and making a precise comparison between intrusion detection systems impossible. As public awareness of intrusion detection grew, independent testing of IDS products began.

InfowarCon 1998 provided the first live independent testing of leading ID products, which included Network Flight Recorder and RealSecure [Northcutt and Novak, 2000]. The companies that created the systems were not happy with the results because they felt the tests

were unfairly designed and the results sensationalized. Lincoln Labs of MIT (http://www.ll.mit.edu) stepped in to fill the demand for impartial and realistic testing of commercial intrusion detection systems in 1999, and their work is still used today as a benchmark.

Lincoln Labs established several key guidelines for comparing ID products. Their most important move was to provide a complete test network, instead of small clips or individual TCP/IP packets, for their test [Northcutt and Novak, 2000]. Intrusion detection systems require a realistic environment in order to measure how they function in the "real world." The tests also require multiple sources of data - after all, not every IDS relies solely on Tcpdump log files. In addition, all but a few corporate networks rely on both Linux and Windows computers, both as servers and as desktops. Therefore it is also important to test intrusion detection systems in a heterogeneous environment, complete with a wide variety of operating system types and versions. Lincoln Labs also added an extra data set free of intrusion activity, so that anomaly detection systems can determine how the test network "normally" behaves before entering the testing phase [Northcutt and Novak, 2000]. All of these factors combine to create a much more complex testing environment than any previous IDS comparisons had used before. The Lincoln Labs tests allow everyone to see how intrusion detection systems perform in real-world situations. While Lincoln Labs made several noteworthy improvements in the field of IDS comparison testing, three key problems remain un-addressed. First, intrusion detection system comparisons demand too precise identification of the test attacks. The second problem is that the tests decide an IDS has correctly identified an attack only if the intrusion alert also reports the exact time of attack.

The last key problem un-addressed by IDS comparison tests is the total lack of "novel" attacks in their test data.

Problem 1: Imprecise Attack Identification

One of the greatest problems that communicating security professionals have had to deal with is labeling the encountered attack methods. Everyone who found a specific attack form would give it a new name, and there were no commonly accepted or official sources to turn to for naming the attacks. Thanks to the work of many groups, including CERT (Computer Emergency Response Team, from Carnegie Mellon University) and BugTraq (http://www.securityfocus.com), this problem has begun to disappear. Since then, a new problem has risen in the realm of IDS comparison to take its place.

When intrusion detection systems send an attack alert during an IDS comparison test, they receive credit only if they use the proper name for the attack being detected. Whether the proper name is a precise CERT vulnerability number, or a 'common use' name like "UDP Flood DoS" varies for each IDS comparison test, and only systems whose alerts report the proper name receive any credit for the attack detection. Transmitting alerts by CERT vulnerability number is not the only way to detect an attack, nor is it the only method of attack reporting that intrusion detection systems use. The FIRE system developed at Iowa State, for example, reports the category of the attack, rather than the specific attack [Dickerson and Dickerson, 2000]. Using FIRE, an intrusion alert would state the presence of an installed backdoor, rather than state whether the backdoor was an installed package like SubSeven (http://www.subseven.ws), or something simpler like netcat. This attack reporting method allows the FIRE research lab to focus on general detection methods instead of

focusing on precisely identifying already well-known exploits. A more detailed alert is more useful to system administrators than a simple one, but any accurate alert is better than no alert at all. Unfortunately, in even the most fair and realistic IDS tests so far, products that give less exact information in their alerts would receive no credit at all. As long as IDS comparison tests maintain rules like this, the tests will continue to favor string-matching misuse systems over more generalized intrusion detection products.

Problem 2: Identifying Time of Attack

The attack-time identification problem is essentially a duplicate of the attack-naming problem: as IDS comparison tests evolved, they became stricter regarding identifying time of attack. To receive credit on the IDS tests, one often must identify when an attack occurred to the second. Even the most lenient of intrusion detection tests only give credit for determining the exact minute of attack. While many systems are capable of this, there is no pressing need for them to do so. Companies do not keep security personnel with a finger directly over a button, able to react within a second of getting an intrusion alert. Further, a forensics team will gather the finest details sometime after the event. Since the function of intrusion detection systems is to serve as an early-warning system, there is no need for an IDS to identify the precise faulty IP packets or the precise second of the attack, unless the intrusion detection system can, without human involvement, respond to the attack.

Any IDS setup that involves an automatic response has two basic options. The first option is to block the source of danger by modifying firewalls or disabling user accounts, and the second option is to use attack scripts against the source of the danger. Malicious users can abuse the former option to perform denial of service at will, and can abuse the latter

option to cause the IDS to attack any targets whose IP addresses they can spoof. If a company's firewall is programmed to automatically block access to any IP address suspected of attacks, then a one person forging IP addresses can cause all major business partners to lose network access in a matter of seconds. Automated response is an extremely risky and potentially legally dangerous route to take. With the exception of automated response, a human must manually perform any action taken due to an alert.

Because the largest delay in the system is almost certainly going to be the human element, getting credit for 'detecting' an attack should not rely on a precise time-of-attack measurement. Systems that analyze data from log files in several-minute blocks of time should still receive credit for intrusion detection. The current IDS tests strongly favor misuse detection systems with simple string-matching methods over the statistically based anomaly detection systems, which often do not use split-second granularity in their intrusion detection. A proper IDS comparison test must be equally fair to both misuse-based and anomaly-based systems; if anything, such a test should favor anomaly-based systems or hybrid systems, as they currently hold the most promise for future advancements in the field of intrusion detection.

Problem 3: Novel Attack Detection

As new vulnerabilities are discovered, software vendors create patches, and system administrators then apply those patches. System administrators worry most about exploits found in the wild before software vendors have created the necessary patch. Services known to be vulnerable are either taken offline as a preventative measure, or closely monitored for signs of intrusion. Once the patch has been released and applied, system administrators stop

worrying as much about the old vulnerability. Such is not the case in the realm of intrusion detection.

All of the benchmark tests rely on very old, very well known attack forms. Developers of intrusion detection systems have had years to study old exploits and find a way to detect them. Virtually no computer connected to the Internet is vulnerable to benchmark attacks. Yet system administrators use the detection rate of these attacks to measure the 'quality' of every modern IDS. Ideally, an intrusion detection system should provide system administrators warning of attacks that have never been seen before, and should also provide a warning when systems have been compromised. These are the qualities that people desire when they think of intrusion detection, not the ability to detect attacks that were prevalent in 1997 [Ye et al, 2001].

When it comes to passing IDS comparison tests that rely solely on network traffic logs for data, Snort may be the ideal software to use. However, Snort is simply a rules-based string matching IDS. It could never detect a novel intrusion method unless one of the previously written rules coincidentally matched with the new attack. Snort would receive higher benchmark scores than a more generalized anomaly-based detection system, even if the anomaly detection system can analyze variances in inbound and outbound traffic load for subtle signs of intrusion, such as the theft of data. Therefore, even though IDS comparison tests have evolved nicely, and now simulate real networks and simulate complete attacks as well as fragments of attacks, these comparison tests still need more development.

Guidelines for Better IDS Tests

While there are three areas that need to be improved upon, it is important for all future IDS comparison tests to duplicate the areas that have already been done well. The well-done areas include the following:

- Include at least two weeks of attack-free data, so that systems can determine what "normal" network activity looks like.
- Include multiple sources of data. Not all systems use the same sources, and no single source can provide enough detail for detecting every possible attack.
- Spread the test attacks out over a period of several days or weeks, rather than launch fifty attacks at a network in fifty minutes.
- Use a variety of attacks, so that missing one possible attack isn't the difference between passing or failing the IDS test.
- Launch a complete, sophisticated attack rather than merely launching one or two disconnected fragments of an attack. Have the attack begin with reconnaissance and ending with theft of resources.

The first two problems described earlier, naming attacks and identifying time of attacks, can be resolved easily. Credit should be given to any IDS that can precisely identify the test attack used. In addition, partial credit should be given to any IDS that can classify the attack into its basic attack phase, such as Information Gathering or Theft of Resources. Full credit should be given to any IDS that identifies the time of attack to within a few seconds. The majority of credit should still be given to any IDS that identifies within a

minute when an attack occurs, and partial credit should be given to any IDS that can pinpoint time of attack within a reasonable block of time, such as five or ten minutes.

The last problem, that no intrusion detection comparison test uses novel attacks to test, has a deceptively simple solution. Rather than try to select a few of the well-known attacks for the IDS test, simulate novel attacks by custom building vulnerabilities and exploits. For example, IDS testers could modify the Apache web server's source code to include an easily exploitable string buffer vulnerability [Oh, 2001].

Part of the IDS comparison could consist of a complete search for and exploitation of that weakness. The simulated hacker could scan for Apache web servers, launch the exploit code, then do what hackers do: install a backdoor, steal data, and use the systems as stepping stones for further attacks. Another part of the IDS comparison could consist of a simulated Denial of Service by modifying a network service's source code to shut down upon receiving a specific, unusual input string.

Some might consider this an unfair test, as it requires detecting attacks that could not possibly be predicted or prepared for ahead of time. While it is likely that no product would be able to reliably pinpoint these novel buffer overflow attacks, any anomaly-based detection system should be able to alert system administrators to the presence of a backdoor, as well as the sudden change in data transfers and login activity. Additionally, if the IDS products are to have any chance of being useful outside of theoretical test labs, then the products will have to be able to detect what has never been seen before.

KNOWLEDGE SHARING FOR INTRUSION DETECTION

Dozens of independent IDS research teams are actively publishing new papers and security utilities. Many of them focus on a particular sub-problem of the field, or on applying one particular problem solving technique. A few research teams try to design a comprehensive system for intrusion detection, with the lofty goal of creating the only tool system administrators need for security [Neumann and Porras, 1999].

Given the diversity of choices available for operating systems, network configuration, security needs, and personal preferences, no one intrusion detection product can outperform and replace all other ID products in every situation. As a result, many companies choose to use multiple vendors' products to provide a more comprehensive detection capability. The combined intrusion detection products can be considered a "hybrid" IDS unique to that particular company. This hybrid IDS detects any attack that any component IDS detects, but has a false alarm rate equal to the sum of all of its components. While the detection rate suggests that such a hybrid system would be ideal for any computer network, practice shows that the high false alarm rate prevents such a system from being useful. Some method must be used to reduce hybrid systems' false alarm rates without compromising their high detection rates.

One issue rarely discussed in IDS research is that of cooperation and knowledge sharing between IDS sensors and systems. Researchers and commercial developers desire to dominate the market rather than share, which means that the advancement of a next-generation, hybridized IDS has not been explored sufficiently. The end result is a series of security tools for system administrators that focus on their own small piece of the puzzle.

The tools produce output data, each in a unique format that requires analysis by a security expert. Unfortunately, because none of these tools share knowledge, they do not take advantage of each other's strengths, nor cover for their weaknesses. It is left to the human element to correlate information between programs such as Scanlogd (http://www.openwall.com/scanlogd), Snort, and log-analyzing utilities.

Most sophisticated attacks are multi-stage in nature, typically occurring over minutes, hours, or even days [Scambray et al, 2001]. These stages categorize the information-gathering steps, as well as the stages where attackers take advantage of the new systems they have compromised (by installing backdoors and stealing data). Trends appear in the order of events that each attack follows. Though attackers have some flexibility in their methods, no system can be compromised until an attacker knows it exists, and no backdoor can be installed until after a system vulnerability is exploited. Logic dictates that an IDS is more likely to be reporting a false positive when it reports an attack like a backdoor if other signs of intrusion are not present.

So far, no one has developed a system to allow very different IDS tools to share knowledge that they learn with other intrusion detection products and sensors. Instead the tasks of eliminating false alarm reports and detecting unnoticed attacks are routinely left to the human element. System administrators are left to consider each attack reported by an intrusion detection tool, and consider whether the alerts all fit together to suggest a multi-stage attack, or if the alert is just a false alarm [Northcutt and Novak, 2000]. There is no reason why this high-level analysis could not be built into an intrusion detection system, instead of relying entirely on the often over-worked human element.

This thesis proposes a methodology on how to reduce IDS sensor information to the core inferred knowledge, and how to share this knowledge with intrusion detection systems in a simple and intelligent manner. The most important knowledge that needs to be shared between sensors ultimately consists of only three elements: attack source, attack destination, and time window of attack. With this knowledge sensors can gather additional forensic evidence of attacks and detect signs of intrusion that would otherwise remain unnoticed. Some IDS sensors do not require all three types of knowledge, while other sensors may not be able to produce all three. Furthermore, the sensors do not need to be fully linked (that is, sensors need not communicate with all other sensors) for the intrusion detection system to function. Outlined below is a protocol for sharing knowledge between sensors, regardless of whether the intrusion detection sensors in a hybrid IDS are made by different companies. A description of the knowledge that can be gleaned from each type of intrusion detection utility is given, in addition to descriptions of the knowledge each type of IDS needs to access. Also included is a discussion on how to take advantage of this knowledge sharing, even though existing IDS tools are designed for competition and not cooperation.

Every IDS uses some pool of data for detection. The most common data sources include network traffic logs, while the most exotic data sources include a user's keyboard typing patterns. Because of the diverse possibility for information sources, it is not possible for all intrusion detection systems to simply share their data. Instead, the data must be filtered, reduced to a simpler form containing the knowledge learned, and then passed on to other detection systems. How can this be accomplished? Consider that all anomaly-based intrusion detection systems, as well as some misuse-based systems rely upon thresholds of some form to determine when an event in the network is declared an attack. These

thresholds are usually adjustable in configuration files to allow security personnel to alter detection rates and false alarm rates. With too many false alarms, an intrusion detection system runs the risk of being turned off and ignored [Northcutt and Novak, 2000]. With too few detections, an IDS fails to provide the protection companies demand. Rather than leave the threshold adjustment to relatively crude corrections made by hand, it is theoretically possible to have an IDS tweak its own threshold levels as knowledge about potential attacks is learned.

A simple protocol for such a knowledge-sharing message system is detailed below. Following the protocol is a description of the knowledge generally gleaned from each phase of a sophisticated attack. Also included are details on how to make existing IDS tools and sensors take part in such a knowledge-sharing scheme, even though the software vendors never considered this knowledge-sharing concept when their products were developed.

Protocol for Knowledge Sharing Messages

Every intrusion detection tool is unique, and every tool also provides a different format for their output. Some tools produce log files that describe intrusion activity in great detail. Some tools send short emails or pages. Others flash a short message to the screen to alert system administrators. Ultimately, however, all these outputs contain at most three fundamental elements: where the attack originates, what the attack targets, and when the attack occurs.

Occasionally, the source of an attack is not clear or is not recorded by an IDS sensor. Often, such as in the case of denial of service attacks, the source of the attack is forged and not particularly useful [Ptacek and Newsham, 2002]. Most of the time, however, the source

IP of intrusive activity (as reported by Scanlogd or Snort, for example) belongs to one of the machines controlled by the attacker. This machine is one the attacker is likely to use again. The ability of sensors to communicate which IP addresses attackers are most likely to use provides a great opportunity for intrusion detection systems to bias activity from these machines over the background noise of the Internet.

The second fundamental element for sensors to share is the attack destination. This consists of two parts: the IP address and the port, or service. This knowledge is sometimes unimportant, such as when Scanlogd reports that a company's entire network was just ping scanned. Occasionally only one of the two parts is relevant: a full port scan of a single IP address, for example, or a network-wide scan for ftp servers. For other situations, both parts together are important. For example an alert of a possible backdoor, including a system IP address and port number, allows other IDS sensors to sift through and check for information leakage from that port.

The final fundamental element consists of a window of time during which the intrusion activity occurs. Scan attempts take place not at one specific moment, but rather over a period of time. The same holds true with information leakage; an intruder stealing data will do so over a period of time, at least until the problem of finite bandwidth is solved. With this knowledge sensors can weight their alert thresholds according to the recency of other signs of attack, and can also focus extra CPU time analyzing specific blocks of time for attack activity.

Given this information, sensors should pass messages in the format shown on Table 1:

Table 1. Message Field Format for Knowledge-Sharing Sensors

Field	**Example**
Sensor	Sensor: Snort
Attack	Attack: Buffer-Overflow
Source	Source: 129.186.13.10
	Source: 129.186.13.*
	Source: 129.186.13.10, 129.186.13.11
Target	Target: 129.186.142.200:22,23
	Target: 129.186.142.*
	Target: 129.186.142.200, 129.186.1.200:22
Starts	Starts: 4/21/02 18:20
Stops	Stops: 4/21/02 18:30

Since not every sensor will be able to capture all three types of information, and many will report multiple IPs, wildcards need to be established in the information sharing protocol. In addition, there is an extra complication from the "Stops" time. IDS sensors can be configured to report either at specified time intervals, or as soon as they detect something. The solution for sensors reporting on time intervals is to report multiple times. For example, an attack might begin at 18:25 and end at 18:35, but the sensor reports at 18:30 and 18:40. In this case, the solution is to send an alert at both time intervals, reporting an attack from 18:25 to 18:30 and another from 18:30 to 18:35. In this manner, the entire attack is reported.

Sensors that are triggered by specific events generally cannot send an alert when an attack stops. This may be a flaw in the sensor's design, but is more likely because the attacks being detected do not have a meaningful duration. An example of such an attack is the buffer overflow, which is as simple as one or two packets. In order to signify attacks with no appreciable duration, set the stop timestamp to the start timestamp.

Attack Phases and Knowledge Gained

Alerts from almost any sensor can be classified into one of these phases. The knowledge that can be reported, as well as its usefulness, depends on the attack phase. This section will describe the knowledge one can expect to obtain from the five attack phases, as well as the relative importance of such knowledge.

Reconnaissance

The knowledge-sharing protocol marks this phase with "Attack: Reconnaissance." Study of the targets information makes the type of reconnaissance readily apparent. A port scan targets a single machine and all ports, resulting in "Target: 129.186.142.200:*" A service scan targets a subnet, but only one port, and looks like "Target: 129.186.142.*:80" A ping sweep would look like a service scan, except without the optional port information included.

The timestamp is not critical for reconnaissance, since time often passes between the first probes of a network and any further attempts to attack the network. Of more use is the source information. The source IP of target machines warn of systems controlled by potentially malicious users, and any further traffic from these IP addresses warrants extra analysis. The more recent and the more thorough the reconnaissance, the longer duration one should more carefully study traffic from an IP.

Scan events are so commonplace on many networks that they draw little attention from system administrators. The trivial importance people place on reconnaissance and information gathering makes it more important for an intrusion detection system to try to sift through these attacks. Some of the knowledge will be faulty, resulting from people 'just

curious' and from cleverly forged source IP addresses. Faulty knowledge will diminish, but not eliminate, an intrusion detection system's ability to use what it has learned. An increase in stored knowledge requires more time to process and compare with new data. With time as particular IP addresses show no additional indications of a threat and as the IDS requires more CPU time to function, the old knowledge can be cleared from memory. By controlling the size of IDS memory, a system administrator can balance stored knowledge against processor usage and detection time.

Information-Gathering

This phase usually follows reconnaissance, and involves a much more diverse range of methods. This step is typically less automated than reconnaissance, so alerts from the information-gathering phase provide system administrators considerably more information. Since this phase normally involves more human interaction on the attacker's part, the timestamp can give intrusion detection systems information about when the attacker is most active. Just as in the previous phase, this phase warns which IPs are controlled by potentially malicious users. The most important knowledge gleaned from detecting information-gathering attempts, though, comes from the Target field.

Even though the reconnaissance phase can be considered an unfocused attack, information gathering by its very nature requires that the attacker focus time and effort gathering minute details about specific aspects of a computer network. The services most likely to be focused on are the services that the attacker either understands the best, or has the newest and most effective exploits for. The systems most carefully scrutinized are the ones that the attacker is most interested in, which can hint at an agenda. After all, the advantages

of attacking an accounting computer are different than the advantages of attacking an employee database. Intrusion detection systems should report alerts with greater ease under three circumstances: if the suspected target was recently involved in an information gathering attack, if the suspected source has a history of information gathering attacks, or both.

Vulnerability Exploitation and Privilege Escalation

The critical knowledge to be gained at the exploitation/escalation stage is a list of systems being targeted. The last two attack phases, particularly theft of resources, are commonly false alarms. Anomaly detection systems have a hard time distinguishing between the flurry of CPU activity and unusual login times that results from a sudden deadline, and the activities of a malicious user. By maintaining a list of systems that have suffered attempted exploits (frequently remote string buffer exploits), an intrusion detection system can greatly reduce the number of false "information leakage" and "theft of resources" alarms.

Backdoor Installation

All knowledge gained from the discovery of an installed backdoor is potentially useful. The time of installation hints that a vulnerability within the system was exploited sometime shortly before the backdoor was detected. The compromised machine should be considered as a possible source of future attacks by all other intrusion detection sensors. Finally, if any part of the intrusion detection system is automatically reactive (even though that might be used in a denial of service attack), then knowing the original source of the

backdoor becomes important, as that is the IP address the system should most likely add to a firewall's block list.

Information Leakage and Theft of Resources

From the intrusion detection system's perspective, this is the last phase of the attack, so it can safely be assumed that the current attack was a fully 'successful' one. The objective for the intrusion detection system should therefore be to prepare additional data for the forensics teams that are probably already on their way. In this case, intrusion detection systems receiving this message can, and should, set their detection thresholds to a more sensitive setting and consider the targeted machine as a potential source for new attacks. Ultimately the most important thing for an IDS to do upon receiving this type of message should be to log as much additional information as possible.

Implementing Knowledge-Sharing

The knowledge-sharing concept outlined above requires two key things for every intrusion detection product. First, every sensor should be capable of producing and transmitting messages of a specific format any time an alert is generated. Second, every IDS should be capable of receiving, interpreting, and using knowledge acquired from other sensors.

Ordinarily, a company would take these functions and build them into an IDS during their product design phase. All intrusion detection tools from the research lab using this knowledge-sharing idea would be compatible with each other, but would not be able to share knowledge with any sensors produced by other companies. This limits the widespread use of

such an intrusion detection system. In order to do well commercially, a company releasing a product that uses this knowledge-sharing design might have to reproduce several tools functionally similar to several of the industry's best intrusion detection utilities.

A better solution would be to use a more layered approach to an intrusion detection system, similar to the TCP/IP protocol. Advances in networking also occur too rapidly for a monolithic approach to the TCP/IP protocol. Instead, a multi-layered TCP stack was designed which would allow certain parts to change independently of other parts of the system. Existing intrusion detection tools, whether simple sensors or complete intrusion detection systems, can be viewed as layer of a more complex system. Depending on who created this layer, it may be proprietary. The components of this layer are almost certainly too complicated for another company to integrate knowledge-sharing into them. Even if some company managed such an integration, what happens when the original companies release upgrades, patches, or entirely new products? Clearly any attempt to integrate the existing, often-specialized intrusion detection tools of today into a hybrid IDS would have to use a multi-layered approach, such as the following:

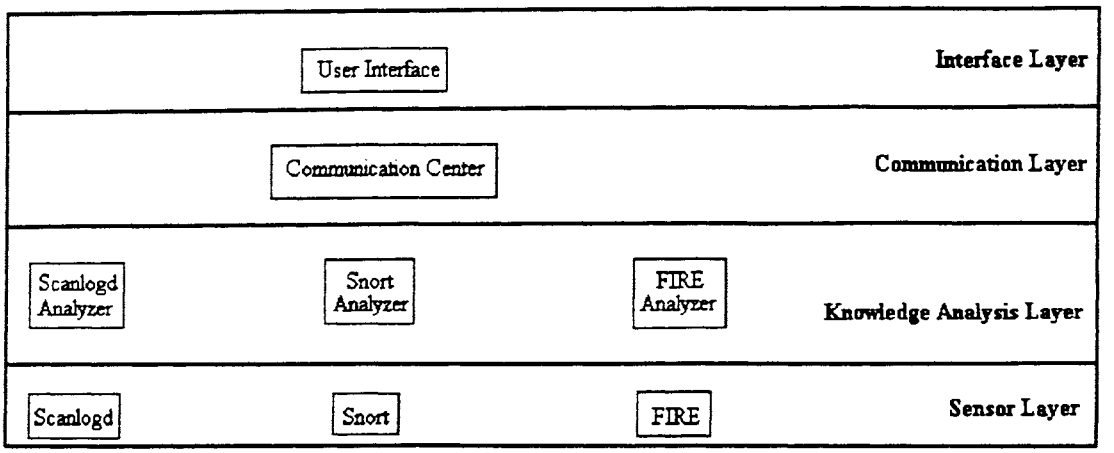

Figure 1. Diagram of the Layered Approach to Knowledge Sharing.

The lowest level layer, the Sensor Layer, is composed of all the intrusion detection utilities to be combined into the knowledge-sharing scheme. Every intrusion detection utility has to generate some sort of output if it is to be of any use. The output, which traditionally consists of intrusion alerts and log files, is passed upward to the next layer. Little can be done with this layer, as the source code to many of the components may be unavailable for modification.

The Knowledge Analysis Layer consists of one module per utility in the Sensor Layer. The purpose of this layer is to sift through the Sensor Layer's output for important knowledge to pass on, and then transmit this knowledge in a properly formatted message. Here is a sample message produced by experimental testing:

Sensor: Scanlogd
Attack: Reconnaissance
Source: 195.49.74.210
Target: 129.186.215.*:22
Starts: 11/23/02 05:40
Stops: 11/23/02 05:40

In the example listed above, Scanlogd reports a scan for SSH (port 22) servers running on the 129.186.215 subnet. Any of a number of intrusion detection products could have detected this scan, but each product has a unique way of reporting the information. The purpose of the Knowledge Analysis Layer is to take reports, convert them into a message format similar to this one, and then transmit it to the Communication Layer.

The Communication Layer serves four purposes. First, it is a central collection point for all knowledge messages. Second, the Communications Layer is the location of the highest-level analysis of possible intrusions, which eliminates redundant alerts, out of date knowledge, and likely false alarms. This function is unnecessary for the purposes of this research, but will become more important as additional components are added to the lower layers. Third, the Communications Layer shares knowledge to the appropriate components in the Knowledge Analysis Layer. The fourth function of this layer is to pass alert information up to the User Interface Layer, and to accept instructions and configuration information from the users.

The last layer is the Interface Layer. This is where intrusion alerts are presented to the users, and where configuration changes can be made to the system. This layer also allows users to request more forensic information from one or more lower-level components in the knowledge-sharing intrusion detection system. The only required function of this layer is to alert users to intrusions, though ideally the interface layer should also allow users to input new knowledge directly into the IDS, as well as delete knowledge known to be faulty or outdated.

AN AAFID COMPARISON

There is one similar concept in the intrusion detection community: Autonomous Agents for Intrusion Detection, or AAFID. AAFID is a product of the CERIAS laboratory at Purdue, and it was "the first architecture that proposed the use of autonomous agents for doing intrusion detection" [Spafford and Zamboni, 2000]. As AAFID also addresses the communication problem that exists in multi-sensor intrusion detection systems, a brief description of AAFID follows, as well as a comparison to the knowledge-sharing idea listed earlier in this paper.

AAFID is written in Perl for UNIX platforms. Windows compatibility testing was underway, but work on the project halted, and there appears to be no plan for continuing research with AAFID. AAFID is designed as a universal method of passing data from various types of IDS sensors upwards to the highest level of the intrusion detection system. AAFID is not an intrusion detection system in and of itself, though the source code includes sample ID sensors that specialize in detecting some specific attacks, like SYN flooding denial of service and password guessing attempts. Instead, AAFID is an architectural design, meant to be programmed into intrusion detection sensors and monitors. The AAFID system allows components of a complex intrusion detection system to pass information in 'messages' from one component to another, even if the other components are on different computers. Using AAFID, control messages can be sent to start and stop components of the system, and data can be passed upward to monitoring components higher in the system hierarchy [Balasubramaniyan et al, 2002].

AAFID meets the first of the two recommendations listed above for the knowledge-sharing concept: that is, sensors built using AAFID can produce and transmit messages any time an alert is generated. The AAFID architecture does not address the other recommendation though: only higher-level components of the AAFID architecture can receive information generated from other IDS components, and this means that the lower level components do not take advantage of any extra knowledge while operating. Higher-level components can pass a message to lower-level components, but the only message that can be passed is a request for sensor data: knowledge cannot be passed downward from a higher level of the AAFID system [Balasubramaniyan et al, 2002].

Also, information is not knowledge. AAFID passes arbitrary messages of strings upward in the sensor net, and by not having a pre-specified message protocol to follow, AAFID ensures that each higher-level component in the system has to learn on its own what the propagated information actually means. If instead bits of underlying knowledge are passed instead, such as the time of an attack, the attacker's potential targets, or the attacker's location, then a major data reduction step has occurred, and it becomes feasible for dozens of different sensors and monitors to make efficient use of the transmitted messages. The AAFID system also does not address the possibility of receiving messages with false conclusions in them, while both the Communication and Interface layers of the knowledge-sharing concept can remove faulty knowledge from the IDS memory. Unfortunately, the AAFID system does little more than display passed messages to a monitor for system administrators to read and interpret. Therefore the AAFID system will have as many false alarms as the sum of all false alarms generated by its sensors, instead of having the reduced

false alarm rate that can be generated by a knowledge-sharing scheme that modifies detection thresholds.

As a final consideration, recall the multi-layered design of the knowledge-sharing scheme. In the multi-layered system design, the most effective intrusion detection tools would be the tools designed with the knowledge sharing protocol in mind, but even the existing IDS tools commonly used in industry could be added to a knowledge sharing system.

For example, consider a knowledge-sharing system using Scanlogd, a misuse-detection utility designed to detect port scans. Scanlogd keeps track of how many ports each IP address attempts to connect to within a specified period of time. If the number of connection attempts exceeds a set threshold, an alert is fired. To combine Scanlogd into a knowledge-sharing system, the "Scanlogd Analyzer" in the Knowledge Analysis Layer would send a properly formatted message up to the Communications Layer, reporting the time of the scan, the source of the scan, and the target IP address and ports of the scan. Should the Scanlogd Analyzer receive known hostile IP addresses from the Communication Layer, then Scanlogd's detection threshold would be lowered for attacks from those particular IP addresses, while staying at the previous threshold for other IP addresses.

For each such tool desired, the 'interpreting' code exists in surrounding layers that manipulate the input and output of each IDS component. This solution may not be elegant from a software engineering point of view, but the end result is that any intrusion detection product today can be added to a knowledge-sharing IDS without requiring complete redesign and recreation. AAFID is a valuable theoretical idea, which shows the possibilities of a distributed intrusion detection system, but proves unrealistic in practice.

EXPERIMENTS

In order to demonstrate the potential of the knowledge-sharing concept outlined above, the following approach was used as a test of knowledge sharing. Ndump (http://www.nightfallsecurity.com/downloads/ndump.html), a program similar in purpose to Tcpdump, was installed on a computer in Iowa State University's Information Systems Security Laboratory (ISSL) network. The computer collected logs of normal network traffic for 96 hours (Monday through Thursday), to be used to determine "normal" network activity. Then over a 24-hour period (Friday), simulated attacks were launched against the ISSL computer network. The data was reduced to a set of metrics that can generalize network traffic patterns. Analysis of the normal data and the intrusion data determined which of the metrics were most important for detecting the attacks used in this experiment. Two intrusion detection systems are then created: one without knowledge sharing, and one with. Both the test and the control IDS contain two separate sensors: Scanlogd (a publicly available intrusion detection tool), and a custom threshold-based anomaly detection system. The control IDS runs the two sensors independently of each other, reporting an alert any time either sensor reports an alert. The control IDS does not share alerts or any other knowledge between its two sensors. The test IDS runs the two sensors using the knowledge-sharing scheme defined earlier in this paper. In the test IDS, both Scanlogd and the anomaly detection system have a default detection threshold, and a more sensitive detection threshold used when indicated by previous knowledge.

In the test IDS, the default thresholds provide a high detection rate while maintaining a tolerable false alarm rate, and the more sensitive thresholds provide a slightly higher detection rate at the cost of an unacceptably high false alarm. For normal time intervals, the former threshold is used for determining intrusions; for time intervals known to include activity from a suspected hacker, the latter threshold is used. For the anomaly detection system of both the test and the control IDS, the first 22 hours of the day containing attacks are used to create the thresholds and determine the initial detection and false alarm rates. The final two hours of attack data are not used in creating the rules, but are instead reserved for additional comparison and validation of the two systems' generated intrusion detection rule-sets.

Test Network

For the experiments, a computer network provided by the Information Systems Security Laboratory was used. This network consists of roughly two dozen computers with a wide variety of hardware platforms, operating systems, and software configurations. The majority of these computers are connected to the Internet, which makes the test environment much more realistic, and makes the test results similar to the results one might find in a real-world network. As a result of using a live network, at several times unusual network traffic appeared in the logs, which required careful scrutiny to determine if it represented an actual attack. The four-day period of "normal" network traffic, which did not include any simulated attacks, still contains unusual data, which simulates the same problem many anomaly detection systems have when they are first used in a real network. Because of the

use of a live network, the detection process becomes more challenging, and is therefore a more accurate and meaningful test of intrusion detection.

Attack Scenario

In order to keep the intrusion detection engines simple, and to keep the number of required data metrics low, attack detection was treated as a binary process. Full credit for detection was assigned for detecting an attack in the proper ten minute time block. Since human interaction and forensic study should be involved in any attack alert, describing the precise method of attack is less important than detecting the attack. Making the output of the IDS a binary value (attacks present versus no attacks present) allows this research to focus solely on the detection process itself, and not get involved in the computer forensics that follows a successful detection.

The following attack scenario consists of the first three of the five phases of attack: Reconnaissance, Information-Gathering, and Vulnerability Exploitation. The attack did not successfully penetrate the network security, so the last two stages (Backdoor Installation and Theft of Resources) are not included in the attack. The attacks mimic the behavior of a common type of malicious computer user: one who wants to break into computers, but does not care too much about which computers are broken into, and therefore does not pursue any one target too thoroughly.

The reconnaissance consists of the following: a ping sweep to locate all active machines, several port scans targeting select computers (each scan run at differing speeds), and service scans of the entire ISSL network for one or two commonly used ports that are

often vulnerable to buffer overflow attacks. Information gathering was performed on the several of the servers to determine operating system and software version information.

The penetration attempts consist of remote password-guessing login attempts. The password guessing was performed "low and slow" – at a rate of no more than three guesses per minute, and less than ten guesses per ten minute increment. Also present are two types of remote buffer overflow attacks: one targets SSH servers, and the other targets email (SMTP). These buffer attacks correspond with the ports used in earlier service scans, mimicking common "script kiddy" behavior of quickly scanning for a specific service and then blindly launching scripts at that service. Most of these attacks are designed to avoid detection, but all of these attacks leave signatures. A properly tuned anomaly detection system should pick up all of the attacks used: the real comparison between systems will be in detection rate versus false alarm rate. The detection process will be a challenging one: anomaly detection systems traditionally have trouble detecting buffer overflow attacks, and this experiment does not include a misuse-detection sensor (such as Snort) that excels at detecting buffer attacks.

Data Metrics

All of the data used for this experiment comes from network traffic logs. On high-traffic networks, storing these logs for any extended period of time becomes exceedingly cost prohibitive, as network traffic logs can easily grow by several gigabytes per day. As an initial step, all incoming web traffic is discarded. Web traffic comprises the majority of the network traffic and would greatly increase the computation time required for analysis of the logs. None of the attacks in this experiment involve web traffic, so the elimination of

network traffic related to web servers will not negatively impact intrusion detection. As a second step, every ten minutes the original log data is processed and reduced down to a statistics log and a connections log to save hard disk space.

The statistics logs contain a series of potentially useful metrics about the packets seen over a ten minute time period. Each metric is a simple frequency count of some aspect of the traffic, such as: the number of destination IP addresses, the number of TCP connections, and the number of unclaimed packets (ones that do not belong to any of the monitored TCP connections). There are several "unique" metrics, which can for example provide information on whether or not a particular destination IP address has been seen in the last two weeks. There are also several "foreign" metrics, which refer to IP addresses outside of the local network being protected. A complete list of the metrics used and a short description of each is listed on Tables 2 through 6.

Table 2. General TCP metrics.

Name of Metric	*Description*
NumConnections	Number of connections seen in a time interval.
NumAttempts	Number of connections attempted (SYN observed).
NumFailed	Number of connections failed (RESET in response to SYN).
NumOpened	Number of connections that were successfully opened.
NumCompleted	Number of connections that were properly completed (normal shutdown sequence).
NumReset	Number of connections that were ended with a RESET.
NumSDPs	Number of SDPs observed.
NumSrcs	Number of source IP addresses observed.
NumDests	Number of destination IP addresses observed.
NumForeign	Number of foreign IP addresses observed.
NumServices	Number of service ports observed in use (successful connections only).
NumServers	Number of hosts that accepted a connection on a service port.
NumUniqSDPs	Number of unique SDPs observed during the interval.
NumUniqSrcs	Number of unique source IP addresses observed during the interval.
NumUniqDests	Number of unique destination IP addresses observed during the interval.
NumUniqForeign	Number of unique foreign SDPs (source IP addresses from outside the subnetwork).
NumUniqServices	Number of unique services observed.
NumUniqServers	Number of unique servers observed.
NumUnclaimed	Number of unclaimed connections observed.

Table 3. TCP service port metrics. These metrics are gathered for each service port observed during each time interval.

Name of Metric	*Description*
NumSrcsForPort[#]	Number of source IP addresses that connected to this port.
NumUniqSrcsForPort[#]	Number of unique source IP addresses that connected to this port (successful or not).
NumDestsForPort[#]	Number of destination IP addresses for this port (successful or not).
NumUniqDestsForPort[#]	Number of unique destination IP addresses (servers) for this port.
NumSDPsForPort[#]	Number of SDPs observed connecting to this port (successful or not).
NumUniqueSDPsForPort[#]	Number of unique SDPs observed connecting to this port (successful or not).
ForeignSDPsForPort[#]	Number of foreign SDPs observed connecting to this port (successful or not).
UniqForeignSDPsForPort[#]	Number of unique foreign SDPs observed connecting to this port (successful or not).
NumServersForPort[#]	Number of hosts observed accepting connections on this port.
NumUniqServersForPort[#]	Number of unique hosts observed accepting connections on this port.
ConnectionsForPort[#]	Number of connections observed on this port (successful or not).
AttemptsForPort[#]	Number of attempted connections on this port (SYN sent).
SuccessesForPort[#]	Number of successful connections on this port.
FailedForPort[#]	Number of failed connections on this port.

Table 4. TCP server metrics. These metrics are gathered for each observed server at each time interval. A server is identified as any host that receives a properly initiated TCP connection from another host.

Name of Metric	*Description*
NumClientsForServer[IP]	Number of clients that successfully connected to this server.
NumUniqClientsForServer[IP]	Number of unique clients that successfully connected to this server.
ForeignClientsForServer[IP]	Number of foreign clients that successfully connected to this server.
UniqForeignClientsForServer[IP]	Number of unique foreign clients that successfully connected to this server.
ConnectionsForServer[IP]	Number of successful connections to this server.

Table 5. General UDP metrics.

Name of Metric	*Description*
UDP-NumConnections	Number of connections seen in a time interval.
UDP-NumFailed	Number of connections failed (an ICMP port unreachable error was recorded).
UDP-NumSDPs	Number of SDPs observed.
UDP-NumSrcs	Number of source IP addresses observed.
UDP-NumDests	Number of destination IP addresses observed.
UDP-NumForeign	Number of foreign IP addresses observed.
UDP-NumServices	Number of service ports observed in use (successful connections only).
UDP-NumServers	Number of hosts that accepted a connection on a service port.
UDP-NumUniqSDPs	Number of unique SDPs observed during the interval.
UDP-NumUniqSrcs	Number of unique source IP addresses observed during the interval.
UDP-NumUniqDests	Number of unique destination IP addresses observed during the interval.
UDP-NumUniqForeign	Number of unique foreign SDPs (source IP addresses from outside the subnetwork).
UDP-NumUniqServices	Number of unique services observed.
UDP-NumUniqServers	Number of unique servers observed.

Table 6. General ICMP metrics.

Name of Metric	*Description*
ICMP-NumConnections	Number of connections seen in a time interval.
ICMP-NumFailed	Number of connections failed (an ICMP port unreachable error was recorded).
ICMP-NumSDTs	Number of SDTs observed.
ICMP-NumSrcs	Number of source IP addresses observed.
ICMP-NumDests	Number of destination IP addresses observed.
ICMP-NumForeign	Number of foreign IP addresses observed.
ICMP-NumTypes	Number of ICMP types observed in use.
ICMP-NumUniqSDTs	Number of unique SDTs observed during the interval.
ICMP-NumUniqSrcs	Number of unique source IP addresses observed during the interval.
ICMP-NumUniqDests	Number of unique destination IP addresses observed during the interval.
ICMP-NumUniqForeign	Number of unique foreign SDTs (source IP addresses from outside the subnetwork).
ICMP-NumUniqTypes	Number of unique ICMP types observed.

The connection logs provide additional information about the TCP and UDP connections that are monitored. The primary use of the connection logs is for the knowledge-sharing part of the experiment: the connection logs record which computer is communication to each other computer, on which ports they talk, how many separate connections they have, and how many packets are sent. This information is important for monitoring specific IP address and ports that are "known" to be potential hackers or targets of potential hackers. The connection logs have the following format:

```
<interval timestamp="09/08/2001 01:30:01">
  <connection sdp="129.186.215.199:129.186.5.112:139" conncount="2"
pktcount="79">;6;6
  <connection sdp="129.186.215.200:129.186.215.199:139" conncount="2"
pktcount="39">;6;3
</interval>
```

For the first connection, the source IP address is 129.186.215.199, and the destination address is 129.186.5.112. The destination port is 139, which means it is probably related to windows file sharing. Over the time interval starting at 1:30AM, two different TCP connections were recorded, which had 79 total packets between the two connections. The final numbers separated by semicolons report the status of the connection. The state values are listed below in Table 7.

Table 7. TCP Connection Status Values and their descriptions.

Status Value	Status of the Associated TCP Connection
3	Still active open connection
6	Successful connection that followed the normal connection shutdown sequence
7	Successful connection that was reset
9	Half-open connection (not yet completed TCP 3-way handshake)
10	Failed connection

The output variable, Attacks, is a binary value that represents whether any intrusion activity occurred during the time interval. If the output has a value of one, it means that one or more intrusive actions occurred. A value of zero means that the time interval contains only normal, safe network traffic. The Attacks variable depends on the various input variables that the ID sensor records. Therefore, the problem of detecting when network intrusions occur depends on being able to use the input variables to classify the Attacks value.

Additional Data Reduction

There are 64 total input variables listed in the five tables above. With five days of network traffic, collected in ten-minute intervals, there are 720 total data patterns. With 64 inputs that range from zero up to the thousands, there is a virtually infinite input space. With such a large input space and only 720 data patterns, each time interval has a unique 64-input pattern. An intrusion detection system could not only learn to memorize which training set patterns correspond to an attack, the IDS could even memorize what timestamp corresponds to each training pattern. The memorization would be most efficiently performed in a simple look-up table; in order to allow the intrusion detection system to make generalizations about attacks instead of just memorize training attacks, the ratio of data patterns to inputs must be greatly increased.

There are entirely too many variables for analysis without several months' worth of traffic logs and hundreds of different attacks to be detected. A simpler anomaly detection system, using fewer inputs, will therefore be used in this experiment. Experience with the attacks used and the type of packets involved in each attack indicates that the most important variables for detecting the experimental intrusive activity will be the TCP variables. This eliminates the 12 ICMP variables and 14 UDP variables from Table 5 and Table 6, bringing the total number of variables down to 38.

The TCP server metrics of Table 4 are created for every server in the network, and each requires separate analysis. The TCP service port metrics of Table 3 are created for every TCP port seen in a time period: as with the TCP server metrics above, the service port metrics would be most effectively analyzed by a separate detection engine per port. The input variables listed in Tables 3 and 4 were designed with high-traffic networks in mind;

since the test network only sees a moderate amount of traffic, these variables can also be eliminated, leaving 19 inputs to consider.

The two most important variables for detecting the Backdoor Installation phase would be NumServers and NumUniqServers. The presence of a new unauthorized server should automatically trigger an intrusion alert. However, the attacks in this experiment do not include the installation of a backdoor, so these two variables are unchanging and unimportant in this experiment, and can be ignored.

Consider for example the use of two of the TCP input variables in an anomaly detection system, NumUnclaimed and NumDests. Along with these two inputs, four more inputs are recommended for anomaly detection: the hourly means of NumUnclaimed and NumDests, and their hourly standard deviations.

Network traffic drastically varies according to time of day: the amount of traffic ultimately depends on the routine of computer users, and human routines are very time-oriented. Furthermore, network traffic is unpredictable and frequently changing. The mean of each variable discussed above cannot be determined, and the means change according to time. The same applies to variable variances as well. Detecting intrusions requires being able to recognize "normal" network traffic. Recognizing normal traffic requires estimating the mean and variance. Determining the sample mean and variance of the four days of background data will not provide an accurate enough estimate. Instead, 24 separate sample means and 24 sample variances shall be calculated, one for each hour of the day. This assumes that while the variables distributions vary with time, the distributions change little when compared to network traffic of previous days at the same hour. Any system administrator will confirm that this is a safe assumption. The distribution of the

NumUnclaimed and NumDests inputs, as well as the values of their means and standard deviations can be found on Figures 2 and 3, and Table 8 below.

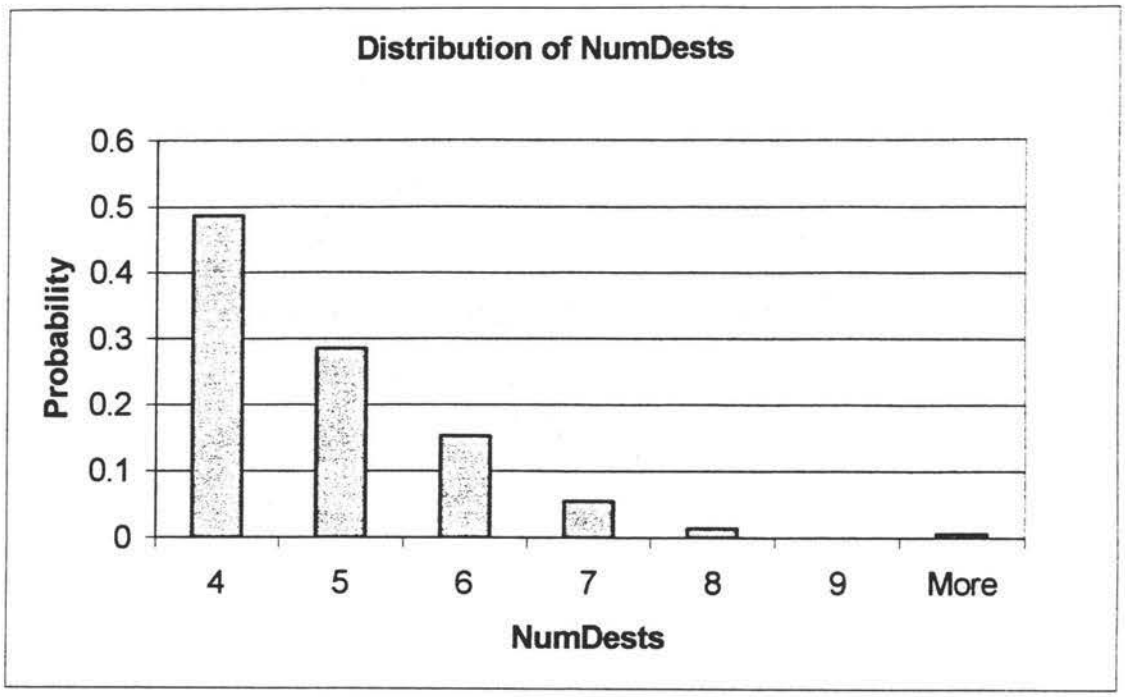

Figure 2. P.D.F. of NumDests variable in the training data set.

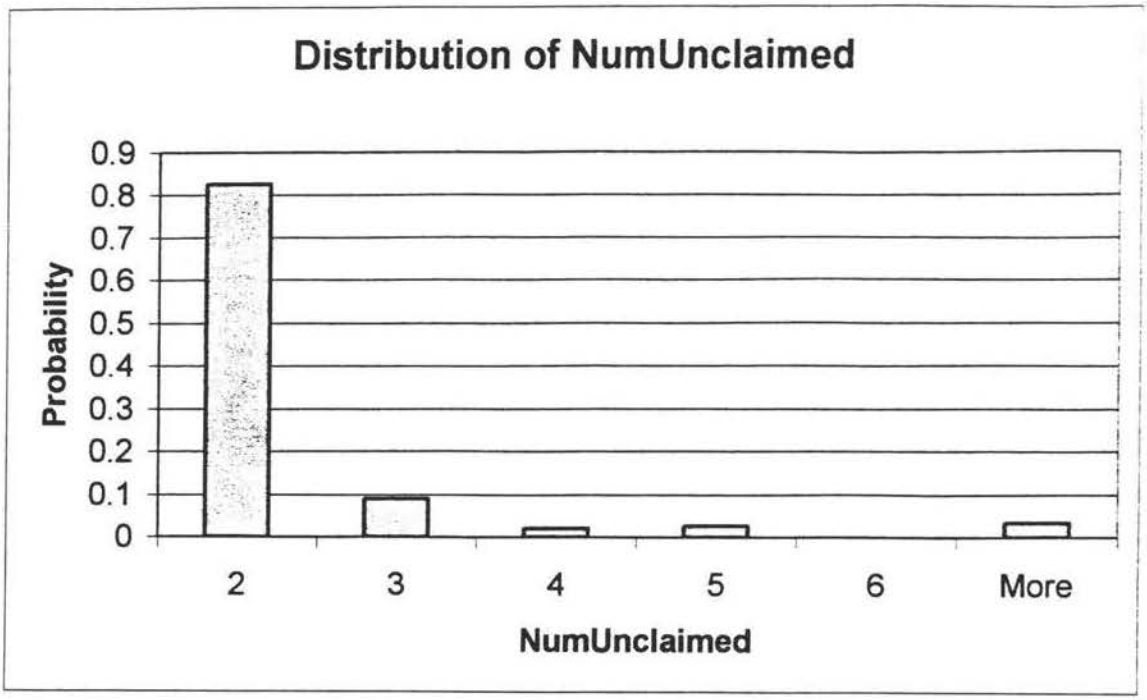

Figure 3. P.D.F. of NumUnclaimed variable in the training data set.

Table 8. Sample means and variances for each hour.

Hour	UnclmAve	UnclmStd	DestsAve	DestsStd
0:00	11.21	16.07	6.08	1.66
1:00	3.83	2.25	5.25	1.42
2:00	2.71	1.06	4.63	0.81
3:00	2.33	0.55	4.17	0.37
4:00	2.25	0.43	4.29	1.02
5:00	2.42	0.76	4.25	0.66
6:00	2.38	0.48	4.25	0.43
7:00	4.79	5.20	4.63	0.86
8:00	5.29	6.80	4.63	0.70
9:00	3.88	4.03	4.79	1.50
10:00	3.33	1.46	5.88	2.71
11:00	4.00	3.34	6.38	0.99
12:00	6.75	9.41	7.54	3.39
13:00	6.29	5.93	6.00	2.02
14:00	4.54	3.15	5.79	1.50
15:00	3.67	3.01	5.75	2.07
16:00	3.83	3.56	5.38	1.70
17:00	3.04	1.06	4.79	0.91
18:00	4.00	6.10	5.25	1.23
19:00	3.58	2.47	6.04	2.68
20:00	2.83	0.99	6.17	2.70
21:00	2.63	1.63	5.54	1.22
22:00	4.33	3.48	5.33	1.21
23:00	3.71	2.03	5.75	1.13

On the training set, 6.8% of the data points have a NumDests value of 7 or higher. If this is used as the threshold for identifying abnormal and therefore intrusive activity, then an intrusion detection system could achieve a 56% detection rate with a 3.3% false alarm rate. Using the NumUnclaimed variable instead, then 7.6% of the data points have a value of 4 or higher. With this threshold, an IDS could achieve a 78% detection rate and a 2.4% false alarm rate with the training set. These detection rates are higher than any of the other 17 TCP metric inputs achieved individually, verifying the earlier theory that the two most

important inputs to use in this experiment were NumUnclaimed and NumDests. These two inputs, their hourly means, and their hourly averages are therefore selected for use in the anomaly detection system used for the knowledge-sharing experimentation.

Results Without Knowledge Sharing

For the control intrusion detection system, which does not use knowledge sharing, all of the alerts reported by Scanlogd's default alert threshold were also reported independently by the anomaly detection system. Scanlogd's default threshold is also tolerant enough that no false alarms are generated. The anomaly detection system does not have a way to report the source IP address or targets of scan attacks like Scanlogd does, but since no knowledge is shared, the control IDS results are the same as the control anomaly detection system alone. Therefore, the rest of this section will only describe the control anomaly detection system and its results.

Anomaly-detection states that all things abnormal indicate an intrusion. This theory has been proven true in past research, although it often leads to a high false alarm rate. One must simply take the input distribution, select the two to five percent of the points that lie furthest from the mean, and declare these the points of possible intrusion. The Denial of Service class attacks often result in a less-active network, but information-gathering techniques and penetration attempts by their very nature require sending more packets, and more unusual packets, not less. Therefore the key is to separate the upper part of the distribution space and declare those to be the intrusions. When analyzing a single variable at a time, this is a simple matter of sorting and testing: to start, demonstrate statistical anomaly-detection using only one variable at a time.

On the training set, 6.8% of the data points have a NumDests value of 7 or higher. If this is used as the threshold for identifying abnormal and therefore intrusive activity, then an intrusion detection system could achieve a 56% detection rate with a 3.3% false alarm rate. To compare these results with the knowledge-sharing method and the other stochastic detection rules, the threshold is used on the test set as well. This results in a 67% detection rate and a 0% false alarm rate. Now the same steps are repeated using the other variable by itself.

Using the NumUnclaimed variable instead, then 7.6% of the data points have a value of 4 or higher. With this threshold, an IDS could achieve a 78% detection rate and a 2.4% false alarm rate with the training set. The test set results turn out the same as with NumDests: 67% detection rate, and a 0% false alarm rate. Clearly the NumUnclaimed variable is the most important single variable in detecting these attacks, which verifies the information theory results discussed earlier in this paper.

Both of these single-variable methods produce reasonably good results, given the data used. The percentage of data points declared to be intrusions varies for the two examples above. This is because of the reasonably low-traffic nature of the testing network. Even a single-point change in the detection threshold results in a large change in pattern classification. But, it is important to note that the above results were achieved by using only a single variable at a time. Using a more complicated multivariable approach to anomaly detection, an intrusion detection system (without knowledge sharing) produces even better results.

Table 9 shows the joint distribution of the training data set, using the two most important input variables: NumDests and NumUnclaimed. This table also demonstrates the

truth behind the "anomalies are intrusions" theory that is so popular among security experts. Over 70% of the training data set has a NumUnclaimed value of 2, and a NumDests value of 4 or 5. Not even one of the simulated attacks found in the training set occurs in this input range. In fact, Table 10 shows the joint probability density function for the attacks: between the two tables, it is readily apparent that the infrequently occurring events are usually intrusions.

Table 9. Joint distribution of all training data patterns.

		Unclm		
		2	**3**	**4+**
Dests	**4**	49.24%	0.00%	0.00%
	5	21.21%	5.30%	0.76%
	6	9.85%	4.55%	2.27%
	7+	2.27%	0.00%	4.55%

Table 10. Joint probability density function of training data attacks.

		Unclm		
		2	**3**	**4+**
Dests	**4**	0.00%	0.00%	0.00%
	5	0.00%	0.00%	11.11%
	6	0.00%	11.11%	22.22%
	7+	11.11%	0.00%	44.44%

Using these two tables, the following decision tree can be formulated to ensure 100% intrusion detection in the training set:

If the NumUnclaimed has a value of 4+ declare an intrusion;
Else, if NumUnclaimed has a value of 3, and NumDests is 6+ declare intrusion;
Else, if NumDests has a value of 7+ declare intrusion;
Else, declare no intrusion occurred.

This decision tree leads to an 8.9% false alarm rate, and a 100% detection rate, in the training set. In the test set, this is a 67% detection rate and a 0% false alarm rate. Given the small size of the test set, these results are excellent. However, this method does not consider the changes with time to the mean and variance of each variable. This means that the decision will not apply to other networks, and may not be accurate on future tests of this same network.

A more flexible intrusion detection method, which may be applicable to other networks instead of just one data set, involves tracking when NumUnclaimed and NumDests are not within one or two standard deviations of the hourly sample mean. 56% of the intrusions can be detected by noticing when NumUnclaimed is more than one standard deviation away from the sample mean, and this elementary technique causes only a 1.6% false alarm rate. Using the same technique on NumDests results in a 44% detection rate with an unacceptably high 19% false alarm rate, so the threshold for NumDests should be set to two standard deviations instead of one. The following generalized decision tree is created:

If NumUnclaimed is greater than one standard deviation from its mean, declare an intrusion;
If NumDests is greater than two standard deviations from its mean, declare an intrusion;
Else, declare no intrusion occurred.

The generalized decision tree leads to an 89% detection rate, with a 3.0% false alarm rate, in the training set. The results for the test set are absolutely horrid: 0% detection rate, and 0% false alarm rate. With a test set consisting of only 3 simulated attacks, the test set results are only useful for comparing different intrusion detection methods included in this

one experiment. The test set results do not look promising for the control intrusion detection system, however.

Results With Knowledge Sharing

While the control IDS uses only the default detection threshold, the test IDS depends on two detection thresholds per sensor. The knowledge-sharing detection system is designed to use two thresholds. The first threshold is used for default situations, which maintains a low false alarm rate while detecting as many of the attacks as possible. The second threshold is reserved for time periods where "known" hostile IP addresses are active (as seen by the connection logs). This second threshold is highly sensitive to detecting attacks, but has a false alarm rate too high to be used all the time: the second threshold could be considered a digital allergy to the identified intruder antigen.

The Scanlogd default threshold for the test IDS is the same as the default threshold used by the control IDS. The more sensitive threshold is capable of detecting slower, stealthier scans, and this threshold is used for all suspicious IP addresses once they are learned to be suspicious. When a scan attack is detected, the Scanlogd Analyzer passes knowledge of the attack source and destination to the communication layer. The communication layer than passes the knowledge downward to the anomaly-detection sensor.

As a default threshold for the test IDS's anomaly detection sensor, the generalized decision tree from the control IDS shall be used:

If NumUnclaimed is greater than one standard deviation from its mean, declare an intrusion;
If NumDests is greater than two standard deviations from its mean, declare an intrusion;
Else, declare no intrusion occurred.

As a second threshold, a more sensitive version of an earlier decision tree is generated, which will detect more attacks, but therefore also trigger more false alarms as well.

If NumDests has a value of six or greater, declare an intrusion;
If NumUnclaimed has a value of three or greater, declare an intrusion;
Else, declare no intrusion occurred even with the suspicious IP address present.

If the second threshold were used exclusively on the training data, the result is 100% attack detection at the cost of a 29.5% false alarm rate. Clearly this threshold is entirely too sensitive to anomalies to be used routinely. However, upon combining both thresholds together under a knowledge-sharing rule, the results are markedly improved.

The very first attack in the training data has a NumUnclaimed value of 10, and the default rule triggers upon it. Scanlogd's alert file contains more knowledge than the binary "attack" output from the anomaly detection system: in particular, the time of the attack, and the source IP address of the scanner is learned. This causes a shared-knowledge message to be transmitted and used by the intrusion detection system: a reconnaissance-type attack occurred at 12:10AM, Monday November 19th. The message also contains the IP address of the would-be intruder, and from this point, all time periods with connections involving that IP address shall be analyzed using the hypersensitive thresholds instead of the default detection thresholds.

Two different IP addresses appear as the source of attacks on Friday, November 23: 64.113.72.241, the most commonly used attack source, as well as 195.49.74.210. Once each IP address can be identified as the source of an attack, the hypersensitive threshold applies to all future time periods with network traffic from these addresses. To make the matter more complicated, not all traffic from these two IP addresses indicates an attack: for example, perfectly normal telnet traffic can be found originating from each IP address in four time periods of the test date.

The test IDS's performance is found by combining the alerts (both correct alerts and false alarms) of both the Scanlogd sensor and the anomaly-detection sensor. The test IDS detected 100% of the attacks in the training set with the same 3% false alarm rate as the default detection system. The results in the test set are also improved: 67% detection rate with 0% false alarm rate, compared to the default detection system's 0% detection and 0% false alarm. As predicted, the detection rate of the knowledge-sharing system (the test IDS) is greater than that of the sum of its parts (the control IDS).

CONCLUSIONS

The ability to compare the performance of different intrusion detection systems is important to both consumers, who want to spend their money on the best security products, and to researchers, who want a way to benchmark their new methods and test their ideas. Given the nature of a capitalistic market, it is unlikely that any intrusion detection system will ever become superior to all other ID tools in every way. In order to defend against the ever-growing, ever-evolving threat of sophisticated attackers, one should use a system of multiple detection tools to provide comprehensive protection from potential attacks.

In the situations tested by these experiments, the overall intrusion detection rate was improved when knowledge of the attackers is shared and used to modify intrusion detection thresholds. False alarm rates in these cases remain steady, and even decrease, due to the knowledge sharing. Given these initial results, it seems safe to conclude that intrusion detection systems would also be improved by sharing knowledge of target systems probed by attackers and the time frames used by each attacker. However, this conclusion remains to be tested.

While the experiments performed do not represent a complete comparison test between the knowledge sharing and the traditional intrusion detection systems, the experimental results do function as a preliminary proof of concept for this method of knowledge sharing. In addition, this research has laid the groundwork for a potentially powerful new line of intrusion detection systems.

This research has demonstrated some of the difficulties of intrusion detection, and pointed out some of the areas left for researchers to explore. Before commercial intrusion

detection systems are likely to adapt a knowledge-sharing method like this one, however, the idea must be expanded and tested further. Here is a list of additional research to consider:

1. Increase the data set size to allow more inputs and more accurate measurement of "normal" network behavior by averaging more than just four days worth of network traffic.
2. Demonstrate the value of the knowledge-sharing concept to host-based intrusion detection systems. So far the concept has only been tested with network-based intrusion data.
3. Expand the scope of the attack simulation and detection scenario to test how knowledge sharing applies to a more complicated problem.
4. Experiment with the 1999 Lincoln Labs IDEVAL data sets, in order to compare a knowledge-sharing system with already proven commercial intrusion detection systems that have also used these data sets. It should be noted that the results would almost certainly be favorable, as all remote attacks in the Lincoln Labs test came from a single IP address.
5. Test combining anomaly and misuse detection systems together with knowledge-sharing methods, then compare the results to more traditional intrusion detection methods.

To summarize, this research serves as an introduction to, and a proof of concept of, a new method of knowledge sharing. Details are also included as to a recommended format for passed messages, as well as a way to use the multi-layered TCP/IP protocol stack as a model for adapting existing intrusion detection tools into a knowledge-sharing scheme. In light of

this research, it can be concluded that knowledge sharing provides an intriguing potential boost to the overall security of computers and computer networks.

REFERENCES

Anonymous, *Maximum Linux Security*, Indianapolis, Indiana: Sams Publishing, 1999

Anonymous, *Maximum Security 2nd Edition.* Indianapolis, Indiana: Sams Publishing, 1998

Balasubramaniyan, J., Garcia-Fernandez, J., Isacoff, D., Spafford, E. and Zamboni, D. "An Architecture for Intrusion Detection Using Autonomous Agents." Electronic source: http://www.cerias.purdue.edu/homes/aafid/docs/tr9805.pdf Downloaded September, 2002

Cherkassky, V. and Mulier, F. *Learning from Data: Concepts, Theory, and Methods.* New York: John Wiley & Sons, Inc, 1998

Cheskin Research and Studio Archetype/Sapient "eCommerce Trust Study." Electronic source: http://www.studioarchetype.com/cheskin/assets/images/etrust.pdf Downloaded: September, 2002

Denning, D. "An Intrusion Detection Model," IEEE Transactions on Software Engineering, Vol. SE-13 No. 2, February 1987

Dickerson, J. E. and Dickerson, J. A. "Fuzzy Network Profiling for Intrusion Detection," Proceedings of NAFIPS 19th International Conference of the North American Fuzzy Information Processing Society, pp 301-306, 2000

Dickerson, J. "Fuzzy Intrusion Detection," Master's thesis, Iowa State University, 2001

Farmer, W., Guttman, J. and Swarup, V. "Security for Mobile Agents: Issues and Requirements," Proceedings of the 19th National Information Systems Security Conference, Vol. 2, pp 591-597, 1996

Frincke, D. "Balancing Cooperation and Risk in Intrusion Detection." Electronic source: http://www.csds.uidaho.edu/director/balcooprisk.pdf Downloaded September, 2002

Garfinkel, S. and Spafford, G. *Practical Unix and Internet Security.* Sebastopol, California: O'Reilly & Associates, Inc., 1996

Gerken, M. "Statistical-Based Intrusion Detection." Electronic source: http://www.sei.cmu.edu/str/descriptions/sbid_body.html Downloaded: October, 2002

Haykin S. *Neural Networks A Comprehensive Foundation.* New York: Macmillan College Publishing Company, 1994

Lane, T. and Brodley, C. "Temporal Sequence Learning and Data Reduction for Anomaly Detection," ACM Transactions on Information and System Security, Vol. 2 No. 3, pp 295-331, August 1999.

Lee, W. and Stolfo, S. "Data Mining Approaches for Intrusion Detection." Electronic source: http://www.usenix.org/publications/library/proceedings/sec98/full_papers/lee/lee.pdf Downloaded June, 2002

Lunt, T. and Jagannathan, R. "A Prototype Real-Time Intrusion-Detection Expert System," Proceedings of the IEEE Symposium on Research in Security and Privacy, pp 59-66, 1998

Mahoney, M. and Chan, P. "Detecting Novel Attacks by Identifying Anomalous Network Packet Headers," Technical Report CS-2001-2. Florida Institute of Technology, 2001

Neumann, P. and Porras, P. "Experience with EMERALD to date," Proceedings of the First USENIX Workshop on Intrusion Detection and Network Monitoring, pp 73-80, 1999.

Nikander, P. and Karvonen, K. "Users and Trust in Cyberspace." Electronic source: http://www.tml.hut.fi/~pnr/publications/cam2000.pdf Downloaded November, 2001

Northcutt S. and Novak J. *Network Intrusion Detection: An Analyst's Handbook.* Indianapolis, Indiana: New Riders Publishing, 2000

Oh, T. "Advanced Buffer Overflow Exploit." Electronic source: http://ohhara.sarang.net/security/adv.txt Downloaded: September, 2001

Porras, P. and Valdes, A. "Live Traffic Analysis of TCP/IP Gateways," ISOC Symposium on Network and Distributed Systems Security, 1998

Ptacek, T. and Newsham, T. "Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection." Electronic source: http://secinf.net/info/ids/idspaper/idspaper.html Downloaded: August, 2002

Ruiu, D. "Cautionary Tales: Stealth Coordinated Attack HOWTO." Electronic source: http://www.nswc.navy.mil/ISSEC/CID/Stealth_Coordinated_Attack.html Downloaded November, 2001

Scambray J., McClure S. and Kurtz G. *Hacking Exposed 2nd Edition.* Osborne: McGraw-Hill, 2001

Snapp, S., Brentano, J., Dias, G., Goan, T., Heberlein, L., Ho, C., Levitt, K., Mukherjee, B., Smaha, S., Grance, T., Teal, D. and Mansur, D. "DIDS (Distributed Intrusion Detection System) – Motivation, Architecture, and an Early Prototype." Electronic source: http://www.silicondefense.com/research/itrex/archive/tracing-papers/snapp91DIDS_prototype.pdf Downloaded: Semptember, 2002

Spafford, E. and Zamboni, D. "Intrusion Detection Using Autonomous Agents," Elsevier Computer Networks 34, pp 547-570, 2000

Sundaram, A. "An Introduction to Intrusion Detection." Electronic source: http://www.acm.org/crossroads/xrds2-4/intrus.html Downloaded: October, 2002

Valdes, A. and Skinner, K. "Adaptive, Model-Based Monitoring for Cyber Attack Detection." Electronic source: http://www.sdl.sri.com/projects/emerald/adaptbn-paper/adaptbn.html Downloaded: August, 2002

Ye N., Li X., Chen Q., Emran S. and Xu M. "Probabilistic Techniques for Intrusion Detection Based on Computer Audit Data," IEEE Transactions on Systems, Man, and Cybernetics Part A, Vol 31, No 4, pp 266-274, July 2001